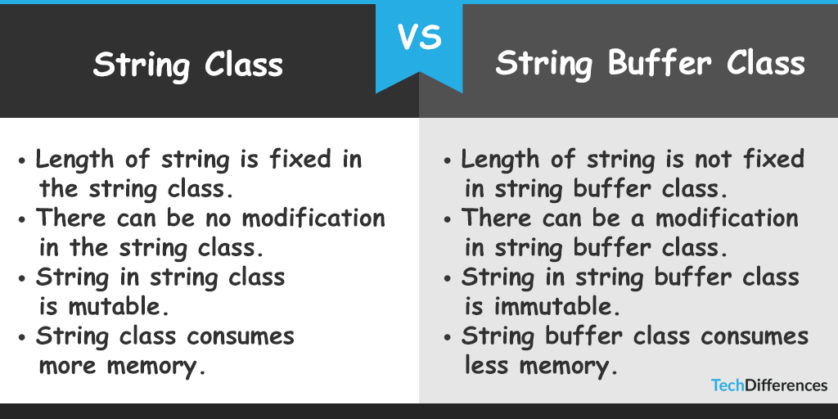
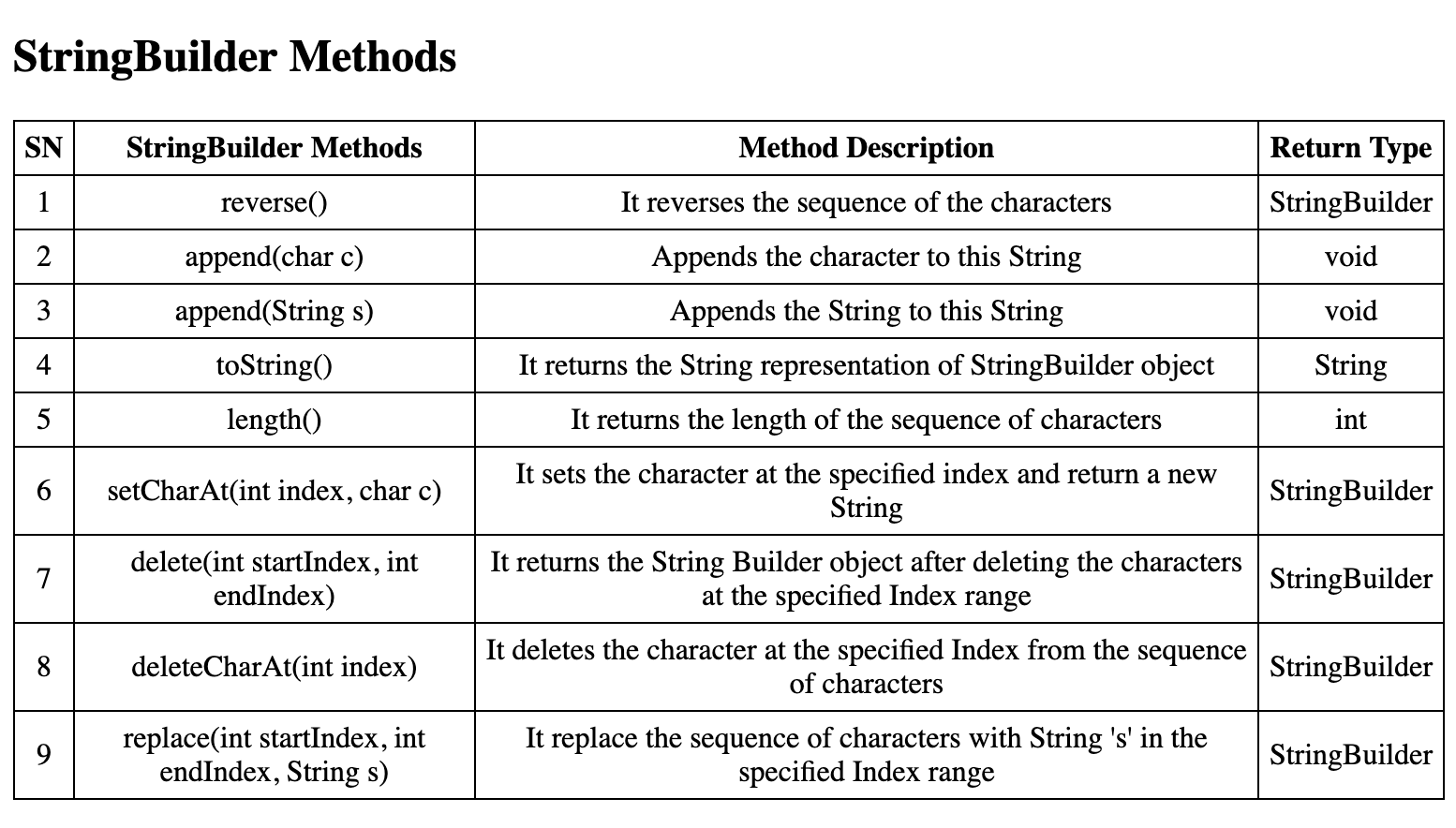
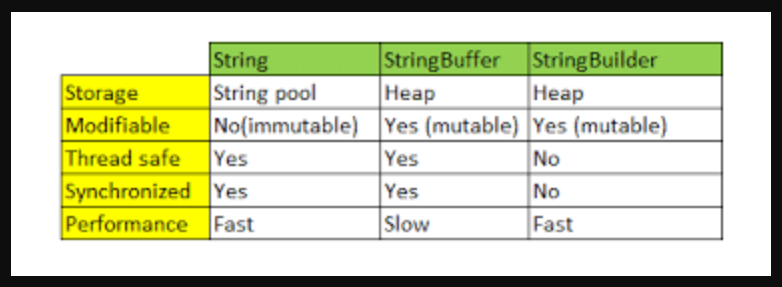


# Java String







In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

**char**[] ch={'j','a','v','a','t','p','o','i','n','t'};  

String s=**new** String(ch);

is same as:

String s="javatpoint";

**Java String** class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

### What is String in java

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

### How to create a string object?

There are two ways to create String object:

1.By string literal

2.By new keyword

### 1.String Literal

Java String literal is created by using double quotes. For Example:

String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool.

**String Pool in Java is a special storage space in Java Heap memory where string literals are stored**. It is also known by the names - String Constant Pool or String Intern Pool. Whenever a string literal is created, the JVM first checks the String Constant Pool before creating a new String object corresponding to it. For

example:

String s1="Welcome";

String s2="Welcome";//It doesn't create a new instance

### Why Java uses the concept of String literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

### 2.By new keyword

String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

## String****Literal vs****String Object

**When we create a String object using the new() operator, it always creates a new object in heap memory. On the other hand, if we create an object using String literal syntax e.g. “Baeldung”, it may return an existing object from the String pool, if it already exists.** Otherwise, it will create a new String object and put in the string pool for future re-use.

At a high level, both are the String objects, but the main difference comes from the point that new() operator always creates a new String object. Also, when we create a String using literal – it is interned.

This will be much more clear when we compare two String objects created using String literal and the new operator:

**String** first = "Baeldung";

**String** second = "Baeldung";

System.out.println(first == second); // TrueCopy

In this example, the String objects will have the same reference.

Next, let's create two different objects using new and check that they have different references:

**String** third = **new** **String**("Baeldung");

**String** fourth = **new** **String**("Baeldung");

System.out.println(third == fourth); // FalseCopy

Similarly, when we compare a String literal with a String object created using new() operator using the == operator, it will return false:

**String** fifth = "Baeldung";

**String** sixth = **new** **String**("Baeldung");

System.out.println(fifth == sixth); // FalseCopy

In general, **we should use the String literal notation when possible**. It is easier to read and it gives the compiler a chance to optimize our code.

**Java String Example**

**public** **class** StringExample{

**public** **static** **void** main(String args[]){

String s1="java";//creating string by java string literal

**char** ch[]={'s','t','r','i','n','g','s'};

String s2=**new** String(ch);//converting char array to string

String s3=**new** String("example");//creating java string by new keyword

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

}}

**Output:**

java

strings

example

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | [char charAt(int index)](https://www.javatpoint.com/java-string-charat) | returns char value for the particular index |
| 2 | [int length()](https://www.javatpoint.com/java-string-length) | returns string length |
| 3 | [String substring(int beginIndex, int endIndex)](https://www.javatpoint.com/java-string-substring) | returns substring for given begin index and end index. |
| 4 | [boolean contains(CharSequence s)](https://www.javatpoint.com/java-string-contains) | returns true or false after matching the sequence of char value. |
| 5 | [boolean equals(Object another)](https://www.javatpoint.com/java-string-equals) | checks the equality of string with the given object. |
| 6 | [boolean isEmpty()](https://www.javatpoint.com/java-string-isempty) | checks if string is empty. |
| 7 | [String concat(String str)](https://www.javatpoint.com/java-string-concat) | concatenates the specified string. |
| 8 | [static String equalsIgnoreCase(String another)](https://www.javatpoint.com/java-string-equalsignorecase) | compares another string. It doesn't check case. |
| 9 | [String[] split(String regex)](https://www.javatpoint.com/java-string-split) | returns a split string matching regex. |
| 10 | [String[] split(String regex, int limit)](https://www.javatpoint.com/java-string-split) | returns a split string matching regex and limit. |
| 11 | [int indexOf(int ch)](https://www.javatpoint.com/java-string-indexof) | returns the specified char value index. |
| 12 | [int indexOf(String substring)](https://www.javatpoint.com/java-string-indexof) | returns the specified substring index. |
| 13 | [int indexOf(String substring, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | returns the specified substring index starting with given index. |
| 14 | [String toLowerCase()](https://www.javatpoint.com/java-string-tolowercase) | returns a string in lowercase. |
| 15 | [String toLowerCase(Locale l)](https://www.javatpoint.com/java-string-tolowercase) | returns a string in lowercase using specified locale. |
| 16 | [String toUpperCase()](https://www.javatpoint.com/java-string-touppercase) | returns a string in uppercase. |
| 17 | [String toUpperCase(Locale l)](https://www.javatpoint.com/java-string-touppercase) | returns a string in uppercase using specified locale. |
| 18 | [String trim()](https://www.javatpoint.com/java-string-trim) | removes beginning and ending spaces of this string. |
| 19 | [static String valueOf(int value)](https://www.javatpoint.com/java-string-valueof) | converts given type into string. It is an overloaded method. |

**The important methods of String class:**

**Java String toUpperCase() and toLowerCase() method**

The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

String s="Sachin";

System.out.println(s.toUpperCase());//SACHIN

System.out.println(s.toLowerCase());//sachin

System.out.println(s);//Sachin(no change in original)

[**Output:**](http://www.javatpoint.com/opr/test.jsp?filename=Testmethodofstringclass)

SACHIN

sachin

Sachin

**Java String trim() method**

The string trim() method eliminates white spaces before and after string.

String s="  Sachin is a cricket player  ";

System.out.println(s);//  Sachin

System.out.println(s.trim());//Sachin

**Output:**

Sachin

Sachin

**Java String startsWith() and endsWith() method**

String s="Sachin";

 System.out.println(s.startsWith("Sa"));//true

 System.out.println(s.endsWith("N"));//false

**Output:**

true

true

**Java String charAt() method**

The string charAt() method returns a character at specified index.

String s="Sachin Ten";

System.out.println(s.charAt(0));//S

System.out.println(s.charAt(3));//h

**Output:**

S

h

**Java String length() method**

The string length() method returns length of the string.

String s="Sachin";

System.out.println(s.length());//6

**Output:**

6

**Java String intern() method**

A pool of strings, initially empty, is maintained privately by the class String.

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

String s=**new** String("Sachin");

String s2=s.intern();

System.out.println(s2);//Sachin

**Output:**

Sachin

**Java String valueOf() method**

The string valueOf() method coverts given type such as int, long, float, double, boolean, char and char array into string.

**int** a=10;

String s=String.valueOf(a);

System.out.println(s+10);

**Output:**

1010

**Java String replace() method**

The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

String s1="Java is a programming language. Java is a platform. Java is an Object oriented and simple language";

String replaceString=s1.replace("Java","Python");//replaces all occurrences of "Java" to "Python"

System.out.println(replaceString);

**Output:**

Kava is a programming language. Kava is a platform. Kava is an Island.

String Concatenation by + (string concatenation) operator

Java string concatenation operator (+) is used to add strings. For Example:

**class** TestStringConcatenation1{

**public** **static** **void** main(String args[]){

   String s="Sachin"+" Tendulkar";

   System.out.println(s);//Sachin Tendulkar

 }

}

**Output:**

Sachin Tendulkar

**String Concatenation by concat() method**

The String concat() method concatenates the specified string to the end of current string. Syntax:

**public** String concat(String another)

Let's see the example of String concat() method.

**class** TestStringConcatenation3{

**public** **static** **void** main(String args[]){

   String s1="Sachin ";

   String s2="Tendulkar";

   String s3=s1.concat(s2);

   System.out.println(s3);//Sachin Tendulkar

  }

}

**Output:**

Sachin Tendulkar

**String compare by equals() method**

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

**public boolean equals(Object another)** compares this string to the specified object.

**public boolean equalsIgnoreCase(String another)** compares this String to another string, ignoring case.

**class** Teststringcomparison1{

**public** **static** **void** main(String args[]){

String s1="Sachin";

String s2=**new** String("Sachin");

String s3="Saurav";

System.***out***.println(s1.equals(s2));//”true” because values are same

System.***out***.println(s1.equals(s3)); //object reference is not same even values are not same for “false”

}

}

**Output:**

true

true

false

**String compare by == operator**

The = = operator compares references not values.

**class** Teststringcomparison3{

**public** **static** **void** main(String args[]){

   String s1="Sachin";

   String s2="Sachin";

   String s3=**new** String("Sachin");

   System.out.println(s1==s2);//true (because both refer to same instance)

   System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)

 }

}

**Output:**

true

false

**String compare by compareTo() method**

The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

**s1 == s2** :0

**s1 > s2**  :positive value (1)

**s1 < s2**  :negative value (-1)

**class** Teststringcomparison4{

**public** **static** **void** main(String args[]){

   String s1="Sachin";

   String s2="Sachin";

   String s3="Ratan";

   System.out.println(s1.compareTo(s2));//0

   System.out.println(s1.compareTo(s3));//1(because s1>s3)

   System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )

 }

}

**Output:**

0

1

-1

## Java String replaceAll() example: replace character

Let's see an example to replace all the occurrences of **a single character**.

**public** **class** ReplaceAllExample1{

**public** **static** **void** main(String args[]){

String s1="java by Edubridge is a very good E-Learning Platform";

String replaceString=s1.replaceAll("a","e");//replaces all occurrences of "a" to "e"

System.out.println(replaceString);

}}

O/P:

jevetpoint is e very good website

## Java String split() method example

The given example returns total number of words in a string excluding space only. It also includes special characters.

**public** **class** SplitExample{

**public** **static** **void** main(String args[]){

String s1="java string split method by javatpoint";

String[] words=s1.split("\\s");//splits the string based on whitespace

//using java foreach loop to print elements of string array

**for**(String w:words){

System.out.println(w);

}

}}

O/P:

java

string

split

method

by

javatpoint

## Java String split() method

**public** **class** SplitExample2{

**public** **static** **void** main(String args[]){

String s1="welcome to split world";

System.out.println("returning words:");

**for**(String w:s1.split("\\s",0)){

System.out.println(w);

}

System.out.println("returning words:");

**for**(String w:s1.split("\\s",1)){

System.out.println(w);

}

System.out.println("returning words:");

**for**(String w:s1.split("\\s",2)){

System.out.println(w);

}

}}

[**O/P:**](http://www.javatpoint.com/opr/test.jsp?filename=SplitExample2)

returning words:

welcome

to

split

world

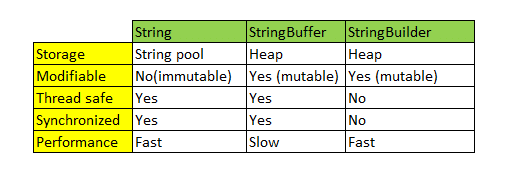
returning words:

welcome to split world

returning words:

welcome

to split world

****

# Java StringBuffer class

Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

|  |
| --- |
| **Note:** Java StringBuffer class is thread safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order. |

**What is mutable string**

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

**StringBuffer append() method**

The append() method concatenates the given argument with this string.

**class** StringBufferExample{

**public** **static** **void** main(String args[]){

StringBuffer sb=**new** StringBuffer("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);

}

}

**Output:**

Hello Java

**StringBuffer insert() method**

The insert() method inserts the given string with this string at the given position.

**class** StringBufferExample2{

**public** **static** **void** main(String args[]){

StringBuffer sb=**new** StringBuffer("Hello ");

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);

}

}

**Output:**

HJavaello

**StringBuffer replace() method**

The replace() method replaces the given string from the specified beginIndex and endIndex.

**class** StringBufferExample3{

**public** **static** **void** main(String args[]){

StringBuffer sb=**new** StringBuffer("Hello");

sb.replace(1,3,"Java");

System.out.println(sb);

}

}

**Output:**

HJavalo

**StringBuffer delete() method**

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

**class** StringBufferExample4{

**public** **static** **void** main(String args[]){

StringBuffer sb=**new** StringBuffer("Hello");

sb.delete(1,3);

System.out.println(sb);

}

}

**Output:**

  Hlo

**StringBuffer reverse() method**

The reverse() method of StringBuilder class reverses the current string.

**class** StringBufferExample5{

**public** **static** **void** main(String args[]){

StringBuffer sb=**new** StringBuffer("Hello");

sb.reverse();

System.out.println(sb);

}

}

**Output:**

 olleH

**Java StringBuilder class**

Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

Let's see the examples of different methods of StringBuilder class.

**StringBuilder append() method**

The StringBuilder append() method concatenates the given argument with this string.

**class** StringBuilderExample{

**public** **static** **void** main(String args[]){

StringBuilder sb=**new** StringBuilder("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);

}

}

**Output:**

Hello Java

**StringBuilder insert() method**

The StringBuilder insert() method inserts the given string with this string at the given position.

**class** StringBuilderExample2{

**public** **static** **void** main(String args[]){

StringBuilder sb=**new** StringBuilder("Hello ");

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);

}

}

**Output:**

HJavaello

**StringBuilder replace() method**

The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.

**class** StringBuilderExample3{

**public** **static** **void** main(String args[]){

StringBuilder sb=**new** StringBuilder("Hello");

sb.replace(1,3,"Java");

System.out.println(sb);

}

}

**Output:**

HJavalo

**StringBuilder delete() method**

The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex.

**class** StringBuilderExample4{

**public** **static** **void** main(String args[]){

StringBuilder sb=**new** StringBuilder("Hello");

sb.delete(1,3);

System.out.println(sb);

}

}

**Output:**

 Hlo

**StringBuilder reverse() method**

The reverse() method of StringBuilder class reverses the current string.

**class** StringBuilderExample5{

**public** **static** **void** main(String args[]){

StringBuilder sb=**new** StringBuilder("Hello");

sb.reverse();

System.out.println(sb);//prints

}

}

**Output:**

olleH

**StringBuffer capacity() method**

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

**class** StringBufferExample6{

**public** **static** **void** main(String args[]){

StringBuffer sb=**new** StringBuffer();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2

}

}

**StringBuffer ensureCapacity() method**

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

**class** StringBufferExample7{

**public** **static** **void** main(String args[]){

StringBuffer sb=**new** StringBuffer();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite languageasdasasfafadadfdaf");

System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2

sb.ensureCapacity(50);//now (34\*2)+2

System.out.println(sb.capacity());//now 70

}

}

**Length()**

StringBuffer sb=**new** StringBuffer();

sb.append("Hello");

System.***out***.println(sb.length()); //5

**charAt()**

StringBuffer sb=**new** StringBuffer();

sb.append("Hello");

System.***out***.println(sb.charAt(1)); //e

**Indexof()**

StringBuffer sb=**new** StringBuffer();

sb.append("Hello");

System.***out***.println(sb.indexOf("l"));

|  |
| --- |
| Performance Test of StringBuffer and StringBuilder Let's see the code to check the performance of StringBuffer and StringBuilder classes.  //Java Program to demonstrate the performance of StringBuffer and StringBuilder classes.  **public** **class** ConcatTest{  **public** **static** **void** main(String[] args){  **long** startTime = System.currentTimeMillis();          StringBuffer sb = **new** StringBuffer("Java");  **for** (**int** i=0; i<10000; i++){              sb.append("edubridge");          }          System.out.println("Time taken by StringBuffer: " + (System.currentTimeMillis() - startTime) + "ms");          startTime = System.currentTimeMillis();          StringBuilder sb2 = **new** StringBuilder("Java");  **for** (**int** i=0; i<10000; i++){              sb2.append("edubridge ");          }          System.out.println("Time taken by StringBuilder: " + (System.currentTimeMillis() - startTime) + "ms");      }  } |